WHAT IS CLAIMED IS:

1. A computer-implemented method of scheduling threads for a plurality of SMT processors, said method comprising:

determining that a first thread in a first run queue is a poor performing thread, wherein the first run queue corresponds to a first SMT processor;

in response to the determination:

writing a first identifier corresponding to the first thread to a second run queue, wherein the second run queue corresponds to a second SMT processor; and

removing the first identifier from the first run queue.

2. The method as described in claim 1 further comprising:

determining that a second thread in the second run queue is another poor performing thread;

in response to the determination regarding the second thread:

writing a second identifier corresponding to the second thread to the first run queue; and

removing the second identifier from the second run queue.

3. The method as described in claim 1, wherein the determination further comprises:

executing a plurality of threads listed in the first run queue, including the first thread, on the first SMT processor, the executing further including:

retrieving a number of cycles value for each thread indicating the number of cycles that occurred while each thread was executing;

retrieving a number of instructions value for each thread indicating the number of instructions that were executed while each thread was executing;

dividing each number of cycles value by its corresponding number of instructions value, the dividing resulting in the CPI value; and

recording the CPI value for each thread in the first queue; and

identifying the first thread as having a CPI value worse than a plurality of the other threads listed in the first run queue.

4. The method as described in claim 3 further comprising:

determining whether each thread was previously moved from one of a plurality of SMT processors' run queues to the first SMT processor's run queue.

5. The method as described in claim 4 further comprising:

determining that the CPI of the first thread has degraded since being moved to the first SMT processor's run queue.

6. The method as described in claim 3 further comprising:

recording the first thread's identifier, the first thread's CPI, and a timestamp to a previously swapped data structure.

7. The method as described in claim 6 wherein the timestamp corresponds to a time selected from the group consisting of a time that the first thread's CPI was calculated, and a time that the first thread's identifier was moved from the first run queue to the second run queue.

8. The method as described in claim 3 further comprising:

averaging the CPI value for each of the plurality of threads with one or more previous CPI values previously calculated for each of the threads, wherein the recorded CPI value includes the average CPI value for each thread.

9. The method as described in claim 1 further comprising:

skipping one or more worse performing threads in comparison to the first thread in response to determining that each of the worse performing threads has previously been moved to the first SMT processor's run queue and each has improved in performance since being moved; and

identifying the first thread after skipping the worse performing threads.

10. The method as described in claim 1 further comprising:

sensing that the first thread is about to complete;

computing a CPI value for the first thread upon its completion;

determining that the CPI value is worse than a threshold value;

in response to the determination, checking whether the first thread was previously moved to the first SMT processor from a previous SMT processor selected from a plurality of SMT processors that includes the first SMT processor and the second SMT processor;

in response to determining that the first thread was previously moved, determining that the first thread's CPI is worse on the first SMT processor than it was on the previous SMT processor; and

wherein the writing and the removing steps are performed in response to determining that the first thread's CPI is worse on the first SMT processor than on the previous SMT processor.

11. An information handling system comprising:

a plurality of SMT processors;

a memory accessible by the processors;

a plurality of software threads and a plurality of run queues stored in the memory, wherein each of the run queues corresponds to one of the SMT processors;

a thread scheduling tool for scheduling threads among the plurality of SMT processors, the thread scheduling tool comprising software code effective to:

determine that a first thread from the plurality of software threads that is in a first run queue selected from the plurality of run queues is a poor performing thread, wherein the first run queue corresponds to a first SMT processor selected from the plurality of SMT processors;

in response to the determination, the software code is effective to:

write a first identifier corresponding to the first thread to a second run queue, wherein the second run queue corresponds to a second SMT processor selected from the plurality of SMT processors; and

remove the first identifier from the first run queue.

12. The information handling system as described in claim 11 wherein the software code is further effective to:

determine that a second thread in the second run queue is another poor performing thread;

in response to the determination regarding the second thread:

write a second identifier corresponding to the second thread to the first run queue; and

remove the second identifier from the second run queue.

13. The information handling system as described in claim 11, wherein the determination further comprises software code effective to:

execute a plurality of threads listed in the first run queue, including the first thread, on the first SMT processor, the execution comprising software code effective to:

retrieve a number of cycles value for each thread indicating the number of cycles that occurred while each thread was executing;

retrieve a number of instructions value for each thread indicating the number of instructions that were executed while each thread was executing;

divide each number of cycles value by its corresponding number of instructions value, the division resulting in the CPI value; and

record the CPI value for each thread in the first queue; and

identify the first thread as having a CPI value worse than a plurality of the other threads listed in the first run queue.

14. The information handling system as described in claim 13 wherein the software code is further effective to:

determine whether each thread was previously moved from one of a plurality of SMT processors' run queues to the first SMT processor's run queue.

15. The information handling system as described in claim 14 wherein the software code is further effective to:

determine that the CPI of the first thread has degraded since being moved to the first SMT processor's run queue.

16. The information handling system as described in claim 13 wherein the software code is further effective to:

record the first thread's identifier, the first thread's CPI, and a timestamp to a previously swapped data structure.

17. The information handling system as described in claim 16 wherein the timestamp corresponds to a time selected from the group consisting of a time that the first thread's CPI was calculated, and a time that the first thread's identifier was moved from the first run queue to the second run queue.

18. The information handling system as described in claim 13 wherein the software code is further effective to:

average the CPI value for each of the plurality of threads with one or more previous CPI values previously calculated for each of the threads, wherein the recorded CPI value includes the average CPI value for each thread.

19. The information handling system as described in claim 10 wherein the software code is further effective to:

skip one or more worse performing threads in comparison to the first thread in response to determining that each of the worse performing threads has previously been moved to the first SMT processor's run queue and each has improved in performance since being moved; and

identify the first thread after skipping the worse performing threads.

20. The information handling system as described in claim 11 wherein the software code is further effective to:

sense that the first thread is about to complete;

compute a CPI value for the first thread upon its completion;

determine that the CPI value is worse than a threshold value;

in response to the determination, check whether the first thread was previously moved to the first SMT processor from a previous SMT processor selected from the plurality of SMT processors that includes the first SMT processor and the second SMT processor;

in response to determining that the first thread was previously moved, determine that the first thread's CPI is worse on the first SMT processor than it was on the previous SMT processor, wherein the software code to write and remove are performed in response to determining that the first thread's CPI is worse on the first SMT processor than on the previous SMT processor.

21. A computer program product stored on a computer operable media for scheduling threads for a plurality of SMT processors, said computer program product comprising:

means for determining that a first thread in a first run queue is a poor performing thread, wherein the first run queue corresponds to a first SMT processor;

in response to the determination:

means for writing a first identifier corresponding to the first thread to a second run queue, wherein the second run queue corresponds to a second SMT processor; and

means for removing the first identifier from the first run queue.

22. The computer program product as described in claim 21 further comprising:

means for determining that a second thread in the second run queue is another poor performing thread;

in response to the determination regarding the second thread:

means for writing a second identifier corresponding to the second thread to the first run queue; and

means for removing the second identifier from the second run queue.

23. The computer program product as described in claim 21, wherein the means for determining further comprises:

means for executing a plurality of threads listed in the first run queue, including the first thread, on the first SMT processor, the means for executing further including:

means for retrieving a number of cycles value for each thread indicating the number of cycles that occurred while each thread was executing;

means for retrieving a number of instructions value for each thread indicating the number of instructions that were executed while each thread was executing;

means for dividing each number of cycles value by its corresponding number of instructions value, the dividing resulting in the CPI value; and

means for recording the CPI value for each thread in the first queue, and

means for identifying the first thread as having a CPI value worse than a plurality of the other threads listed in the first run queue.

24. The computer program product as described in claim 23 further comprising:

means for determining whether each thread was previously moved from one of a plurality of SMT processors' run queues to the first SMT processor's run queue.

25. The computer program product as described in claim 24 further comprising:

means for determining that the CPI of the first thread has degraded since being moved to the first SMT processor's run queue.

26. The computer program product as described in claim 23 further comprising:

means for recording the first thread's identifier, the first thread's CPI, and a timestamp to a previously swapped data structure.

27. The computer program product as described in claim 26 wherein the timestamp corresponds to a time selected from the group consisting of a time that the first thread's CPI was calculated, and a time that the first thread's identifier was moved from the first run queue to the second run queue.

28. The computer program product as described in claim 23 further comprising:

means for averaging the CPI value for each of the plurality of threads with one or more previous CPI values previously calculated for each of the threads, wherein the recorded CPI value includes the average CPI value for each thread.

29. The computer program product as described in claim 21 further comprising:

means for skipping one or more worse performing threads in comparison to the first thread in response to determining that each of the worse performing threads has previously been moved to the first SMT processor's run queue and each has improved in performance since being moved; and

means for identifying the first thread after skipping the worse performing threads.

30. The computer program product as described in claim 21 further comprising:

means for sensing that the first thread is about to complete;

means for computing a CPI value for the first thread upon its completion;

means for determining that the CPI value is worse than a threshold value;

in response to the determination that the CPI value is worse, means for checking whether the first thread was previously moved to the first SMT processor from a previous SMT processor selected from a plurality of SMT processors that includes the first SMT processor and the second SMT processor;

in response to determining that the first thread was previously moved, means for determining that the first thread's CPI is worse on the first SMT processor than it was on the previous SMT processor; and

wherein the means for writing and the means for removing are performed in response to determining that

```
21          the first thread's CPI is worse on the first SMT
22          processor than on the previous SMT processor.
```